Modify PRH Handler Architecture to Handle Delayed Registration in SO

The 5G NG-RAN architecture, which was first specified in 3GPP Release 15, supports slicing and flexible deployment of the RAN building blocks. The NG-RAN radio base stations (known as gNBs) incorporate three main functional modules – the centralized unit (CU), the distributed unit (DU), and the radio unit (RU) or active antenna unit (AAU) – which can be deployed in multiple combinations, according to the mobile operator's requirements and preferences. The CU can be further disaggregated into CU user plane (CU-UP) and CU data plane (CU-DU), accommodating the separation between the control plane and the user plane.

In ONAP, the service model below can be used for deployment of a 5G NG-RAN gNB i.e. CUCP, CUUP, DU as CNF (VF) models, and antennas (RU) as PNF composed inside a service. It is also possible that the DU model is a PNF.

The important point to note here , is that a service model could contain multiple PNFs in a 5G NG-RAN e.g. multiple RUs/DUs.

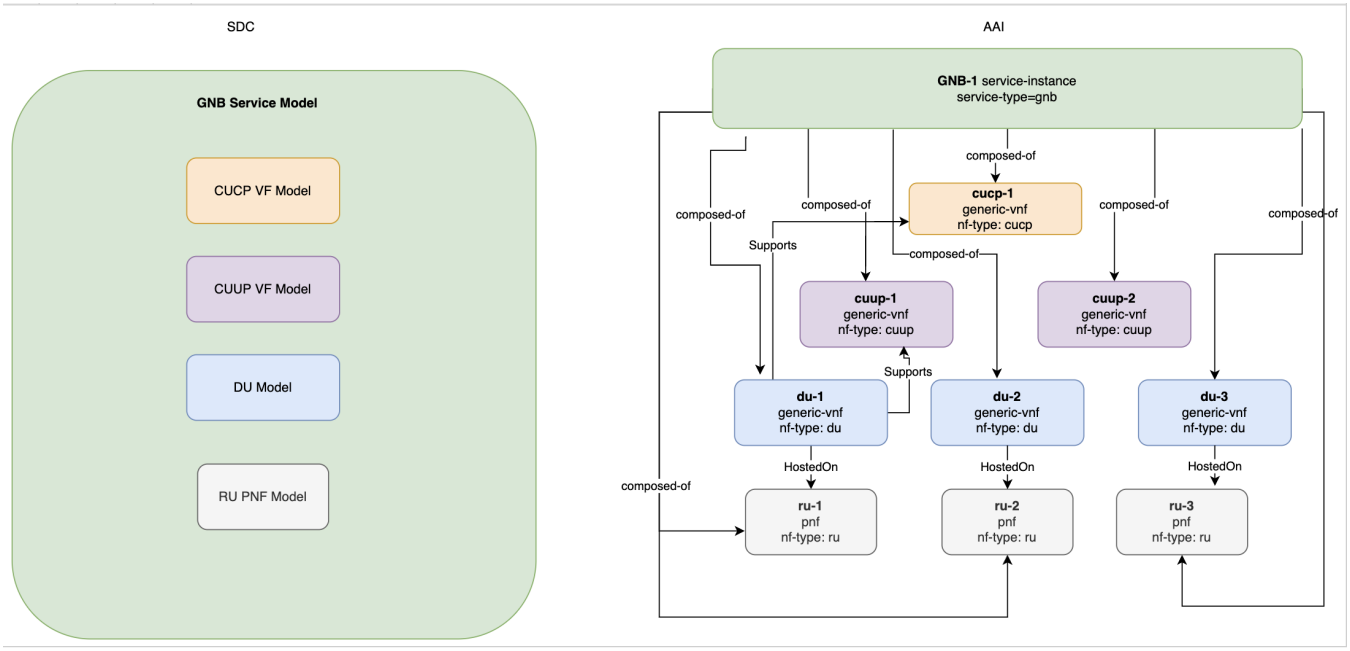


Figure 1. An example 5G NG-RAN service model in ONAP

In gNBs, the CUCP, CUUP and DUs are activated before the RUs. For this reason, we made a change to SO Orchestration Flow table so that all PNFs get registered only after all CNFs in a model are instantiated and activated,, as shown below.

SO Catalog DB, the orchestration_flow_reference table for COMPOSITE_ACTION="Service-Macro-Create" after the sequence update:

id	COMPOSITE_ACTION	SEQ_NO	FLOW_NAME	FLOW_VERSION	NB_REQ_REF_LOOKUP_ID	SCOPE	ACTION
421	Service-Macro-Create	1	AssignServiceInstanceBB	1	94	NULL	NULL
424	Service-Macro-Create	2	CreateNetworkCollectionBB	1	94	NULL	NULL
427	Service-Macro-Create	3	AssignNetworkBB	1	94	NULL	NULL
430	Service-Macro-Create	4	AssignVnfBB	1	94	NULL	NULL
433	Service-Macro-Create	5	AssignVolumeGroupBB	1	94	NULL	NULL
436	Service-Macro-Create	6	AssignVfModuleBB	1	94	NULL	NULL
439	Service-Macro-Create	7	ControllerExecutionBB	1	94	vnf	config-assign
457	Service-Macro-Create	8	CreateNetworkBB	1	94	NULL	NULL
460	Service-Macro-Create	9	ActivateNetworkBB	1	94	NULL	NULL
463	Service-Macro-Create	10	CreateVolumeGroupBB	1	94	NULL	NULL
466	Service-Macro-Create	11	ActivateVolumeGroupBB	1	94	NULL	NULL
469	Service-Macro-Create	12	CreateVfModuleBB	1	94	NULL	NULL
472	Service-Macro-Create	13	ActivateVfModuleBB	1	94	NULL	NULL
475	Service-Macro-Create	14	ControllerExecutionBB	1	94	vnf	config-deploy
478	Service-Macro-Create	15	ActivateVnfBB	1	94	NULL	NULL
481	Service-Macro-Create	16	ActivateNetworkCollectionBB	1	94	NULL	NULL
442	Service-Macro-Create	17	AssignPnfBB	1	94	NULL	NULL
445	Service-Macro-Create	18	WaitForPnfReadyBB	1	94	NULL	NULL
448	Service-Macro-Create	19	ControllerExecutionBB	1	94	pnf	config-assign
451	Service-Macro-Create	20	ControllerExecutionBB	1	94	pnf	config-deploy
454	Service-Macro-Create	21	ActivatePnfBB	1	94	NULL	NULL
484	Service-Macro-Create	22	ActivateServiceInstanceBB	1	94	NULL	NULL

Figure 2: SO Catalog DB, the orchestration_flow_reference table for COMPOSITE_ACTION="Service-Macro-Create" after the sequence update

Problem Statement

When the service model shown in Figure 1 is instantiated using SO Macro Flow for 1 CUCP (cucp-1), 2 CUUPs (cuup-1, cuup-2), 3 DUs(du-1, du-2, du-3), and 3 RUs(ru-1, ru-2, ru-3), the following is observed during the instantiation flow:

1. After the CNFs are activated, ru-1 (the first RU specified in the SO instantiation request) is registered in AAI, and SO waits for the PNF Ready event from PRH handler, as described here: https://docs.onap.org/projects/onap-integration/en/latest/docs_5g_pnf_pnp.html and here: https://docs.onap.org/projects/onap-dcaegen2/en/latest/sections/services/prh/architecture.html
2. After the PNF READY event is received from PRH Handler, SO updates AAI for ru-1, and then starts the same process for ru-2, and after ru-1 is processed in the same way, it goes on to ru-3.
3. PRH Handler sends the PNF READY event to SO, if it receives a PNF Registration Event from the RU. This event is sent when the RU is started /boots up.

The current flow of the PRH handler is shown here (Reference: https://docs.onap.org/projects/onap-dcaegen2/en/latest/sections/services/prh/architecture.html)

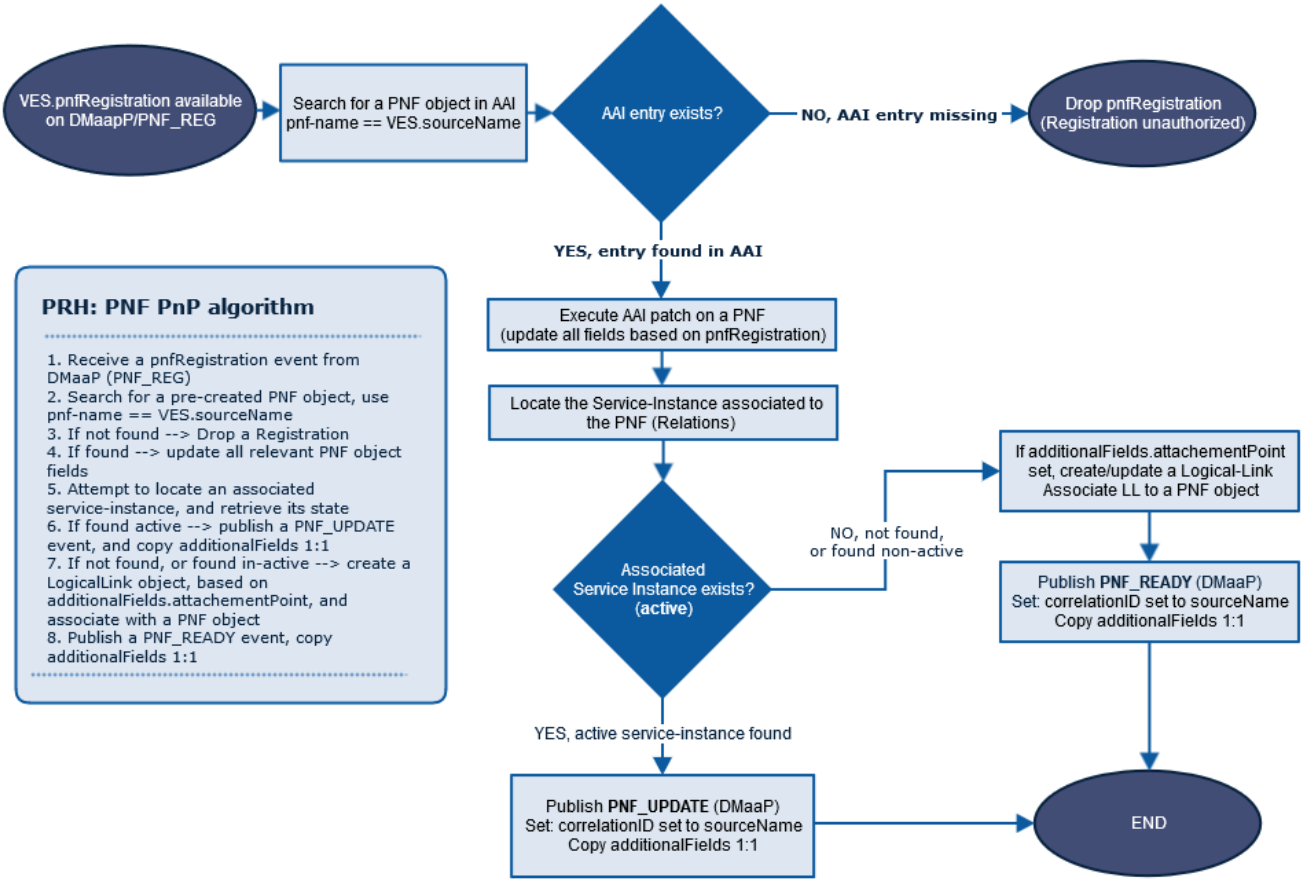


Figure 3. Present PRH Processing Flow

In this scenario, if ru-3 boots up first (before ru-1 and ru-2), then PRH will receive the PNF Registration Event from ru-3, but it **will not find the corresponding AAI PNF object**, since SO would not have created it (since ru-1 and ru-2 have not booted up, and it is till waiting for the PNF READY from them).

Since PRH Handler does not find the AAI object for ru-3 in AAI, **it drops the PNF Registration event for ru-3.**

When SO waits for the event from ru-3, it may never receive it unless ru-3 sends the event again.

Hence an order of RU boot up sequence is enforced by the SO Macro flow.

This is an unnecessary restriction on the bring up procedure of the gNB, and a solution is required to remove this limitation.

Both the solutions proposed below will only be active when there are multiple PNFs inside a service model. For services with a single PNF, the current working functionality will be unaffected.

Proposed Solution - Option 1 - Database Storage

A possible solution is that if the PRH Handler does not find the AAI object for a PNF Registration event, it should store the event in a Database, and poll AAI to check if the object is created. If it finds the object during a poll, the usual steps are followed. A limit could be set on the age of a PNF Registration event, and polling for an event could cease after the age limit is crossed.

The diagram below shows the new flow for PNF Plug and Play, with blocks in blue as new steps. **A new PNF_REG table in the shared Maria DB cluster (used by SO)**, could be used for storage of events. Events are deleted after a certain age limit, or if the AAI PNF object is found, to free up DB storage space.

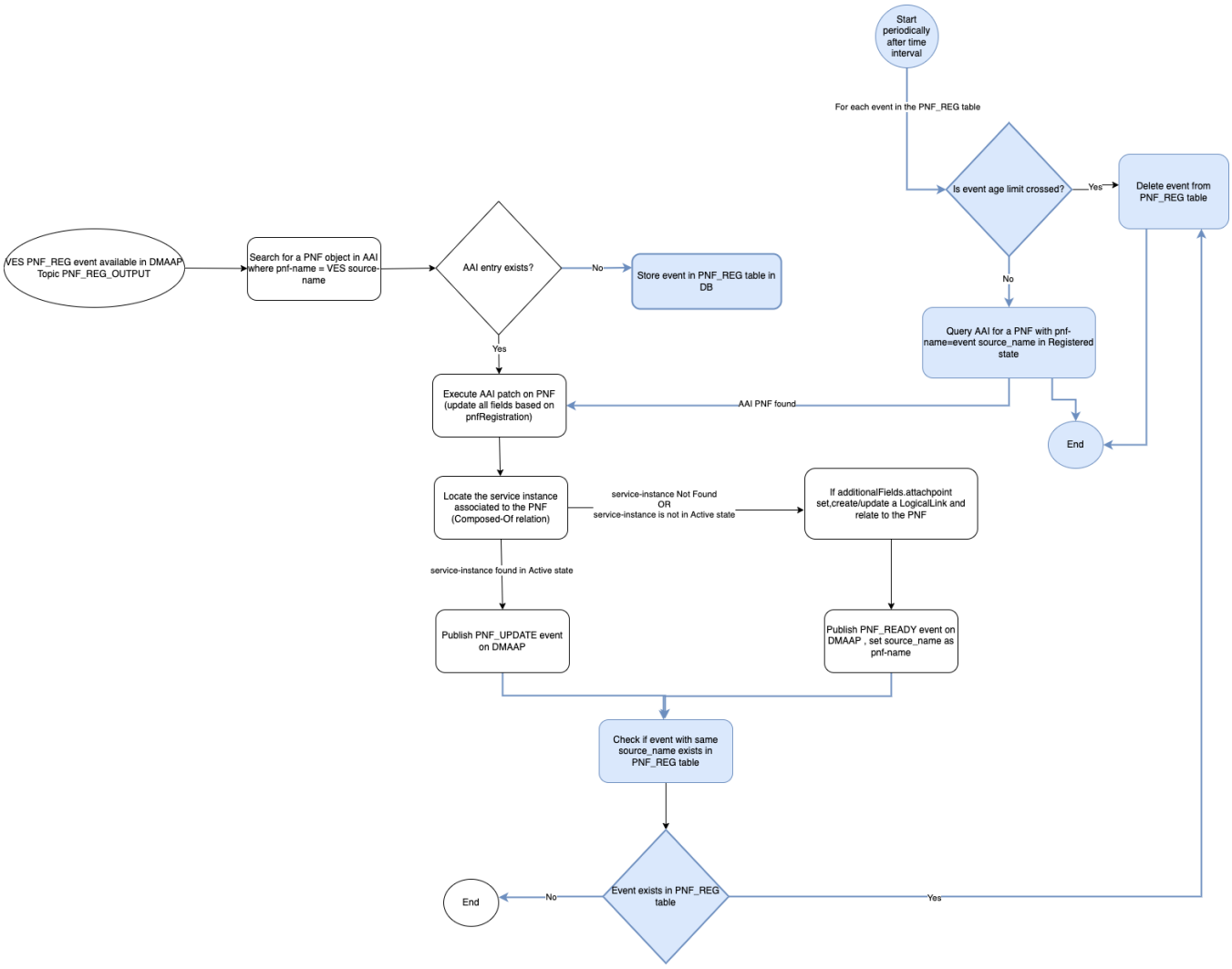


Figure 4. Modified PRH Processing

Proposed Solution - Option 2 - Use Kafka For Event Retention

Another solution is to use Kafka's retention and commit mechanism to access the earlier events that could not be processed by PRH.

Note: We need to use Kafka Client in PRH to use auto-commit option, and not DMAAP REST Service.

Set auto-commit for the consumer group of PRH to False.

PRH will read the set of events, and check if all are processed (found in AAI). If it finds some that are not processed, it does not commit (offset is not moved to last read event and remains in place).

On next read, new events along with old uncommitted events are read. It will again check AAI for each event. If it finds that all events are processed (found in AAI), it will commit.

Next read after commit will get new set of events.

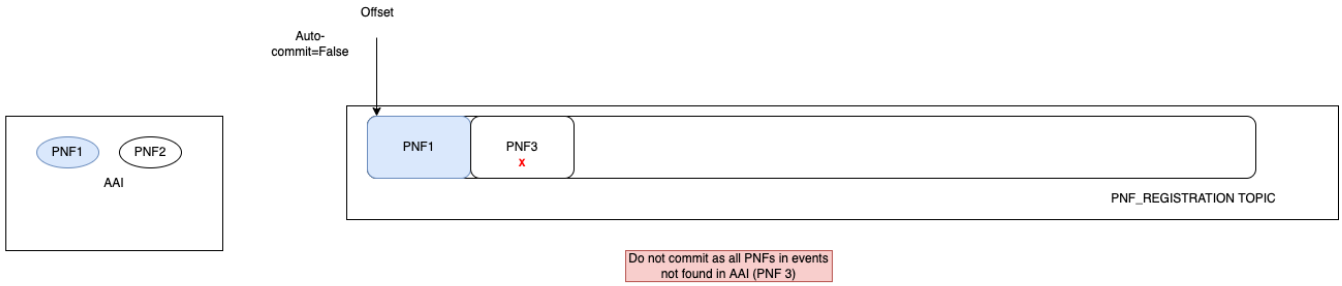
In the diagram below:

1. At start up of PRH, it starts listening on PNF_REGISTRATION topic for events.
2. Two PNF objects are created by SO in AAI, PNF1 and PNF2.
3. First Iteration: (poll)
    a. PRH finds events for PNF 1 and PNF 3.
    b. Commit flag is set to true.
    c. It checks AAI for PNF 1 and finds it. The status of the PNF is not Active and hence, it updates it.

    d. It checks AAI for PNF 3 and does not find it. It checks the age of the event (using creation timestamp). If the age is not more than a limit (e.g. 7 days) , it sets commit flag to false. This indicates that there is a recent PNF event (PNF 3) that is received before its registration in AAI by SO.
    e. Since commit flag is false after reading events, commit is not done by PRH, and the offset does not move.
4. One more PNF object is created by SO in AAI, PNF 3.
5. Second poll:
    a. Commit flag is set to true.
    b. PRH finds events for PNF 1 , PNF 3, and PNF 4.
    c. It checks AAI for PNF 1 and finds it. It finds the status as Active, hence it does not update it.
    d. It checks AAI for PNF 3 and finds it.The status of the PNF is not Active and hence, it updates it.
    e. It checks AAI for PNF 4, and does not find it. It checks the age of the event (using creation timestamp). If the age is not more than a limit (e.g. 7 days) , it sets commit flag to false.This indicates that there is a PNF event (PNF 4) that is received before its registration in AAI by SO.
    f. Since commit flag is false after reading events, commit is not done by PRH, and the offset does not move.
6. One more PNF object is created by SO in AAI, PNF 4.
7. Third poll:
    a. Commit flag is set to true.
    b. PRH finds events for PNF 1 , PNF 3, PNF 4 and PNF 2.
    c. It checks AAI for PNF 1 and finds it. It finds the status as Active, hence it does not update it.
    d. It checks AAI for PNF 3 and finds it.It finds the status as Active, hence it does not update it.
    e. It checks AAI for PNF 4, and does not find it. It checks the age of the event (using creation timestamp). If the age is not more than a limit (e.g. 7 days) , it sets commit flag to false. This indicates that there is a PNF event (PNF 4) that is received before its registration in AAI by SO.
    f. It checks AAI for PNF 2 and finds it.The status of the PNF is not Active and hence, it updates it.
    g. Since commit flag is true, it commits and moves the offset to last read event.
8. Fourth poll:
    a. Commit flag is set to true.
    b. PRH finds events for PNF 1 , PNF 3, PNF 4 and PNF 2.
    c. It checks AAI for PNF 1 and finds it. It finds the status as Active, hence it does not update it.
    d. It checks AAI for PNF 3 and finds it.It finds the status as Active, hence it does not update it.
    e. It checks AAI for PNF 4 and finds it.The status of the PNF is not Active and hence, it updates it.
    f. It checks AAI for PNF 2 and finds it. It finds the status as Active, hence it does not update it.
    g. Since commit flag is true, it commits and moves the offset after last read event (PNF 2)
9. In the next poll, same process is repeated after last read events from new offset i.e. events received after PNF 2
10. To increase processing throughput, PRH pods can be replicated and partitions in the Kafka topic can be increased accordingly - e.g. 2 PRH pods with 2 partitions in the topic. Each pod is a consumer (in the same consumer group). The events will be assigned to the two different partitions in the topic by Kafka, such that each pod receives different events and not duplicate events. Offsets are specific to a partition, hence in this case, each pod would manage its offset in its own partition. The PNF_REGISTRATION topic can be configured at design time to have multiple partitions to enable scaling up of pods to increase consumption throughput.
11. It can be made configurable during installation of PRH or using a configmap - if PRH should operate in the default mode with auto-commit enabled , or if it should disable auto-commit and retry correlation of events, as described above.
12. Since DMAAP MR cannot be used , native Java Apache Kafka Consumer can be used. This change can be made in the DCAE SDK DMAAP Client, also including more granular APIs which allow offset management.

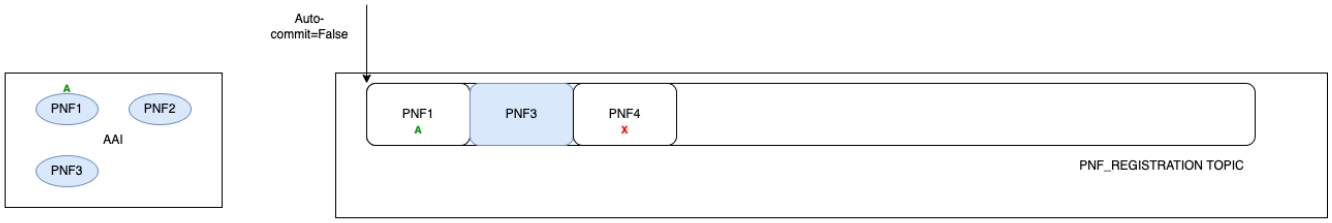
Note: Storing of events in internal memory is avoided, since the pod may be replicated or restarted. Since number of PNF Registration events is not as much as performance or fault events, number of AAI queries will be within limit without affecting performance.

First Poll



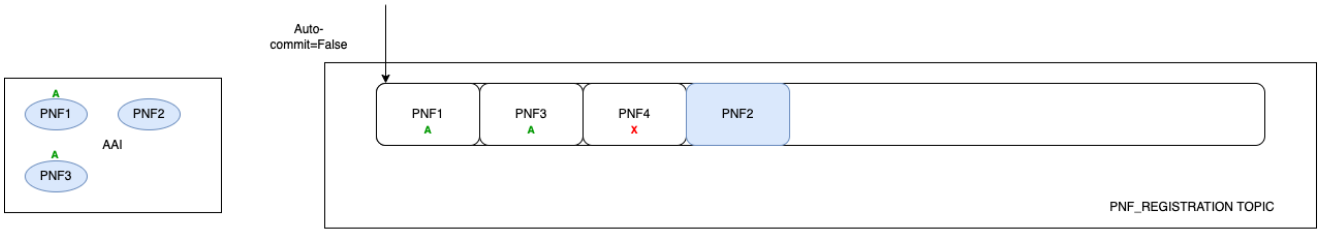
Do not commit as all PNFs in events not found in AAI (PNF 3)

Second Poll



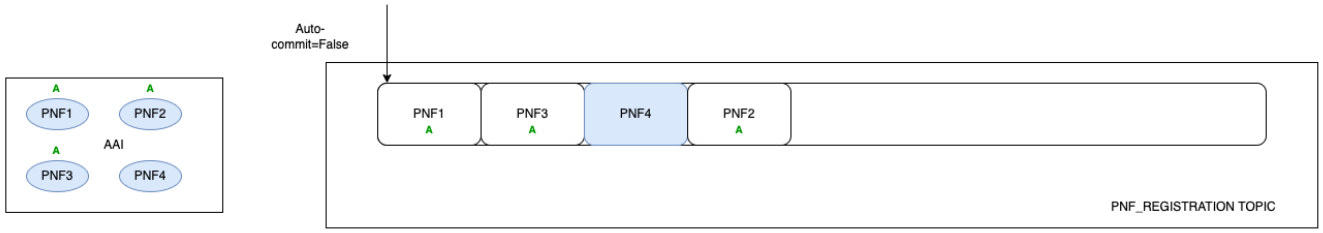
Do not commit as all PNFs in events not found in AAI (PNF 4)

Third Poll

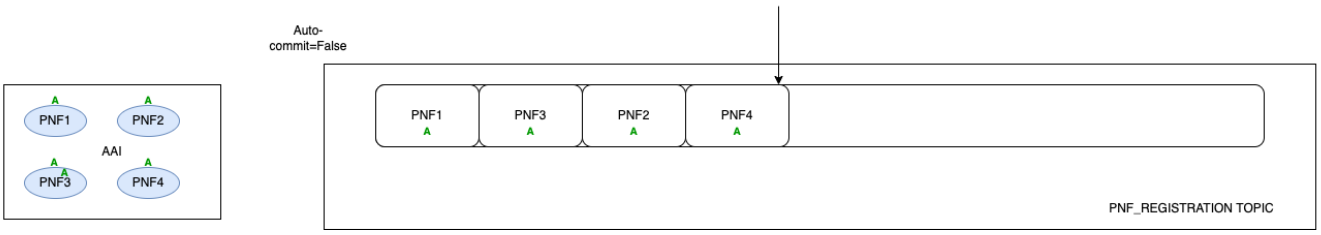


Do not commit as all PNFs in events not found in AAI (PNF 4)

Fourth Poll



Commit (All events till now are found in AAI)


Auto-
commit=False
PNF1
PNF2
AAI
PNF3
PNF4
PNF1
PNF3
PNF2
PNF4
PNF_REGISTRATION TOPIC